中山大學
SUN YAT-SEN UNIVERSITY

国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

计算机图形学

半透明场景渲染

陶钧

taoj23@mail.sysu.edu.cn

中山大学 计算机学院

国家超级计算广州中心

半透明场景渲染

- 场景表示
  - 网格与体数据
- 网格渲染
  - 不透明：深度测试
  - 半透明：blending + order independent transparency
- 体数据渲染
  - Direct volume rendering
- 转换与融合
  - 体数据→网格：marching cubes
  - 网格→体数据：signed distance field (SDF), occupancy field
  - 神经网络渲染

场景渲染

- 光栅化渲染

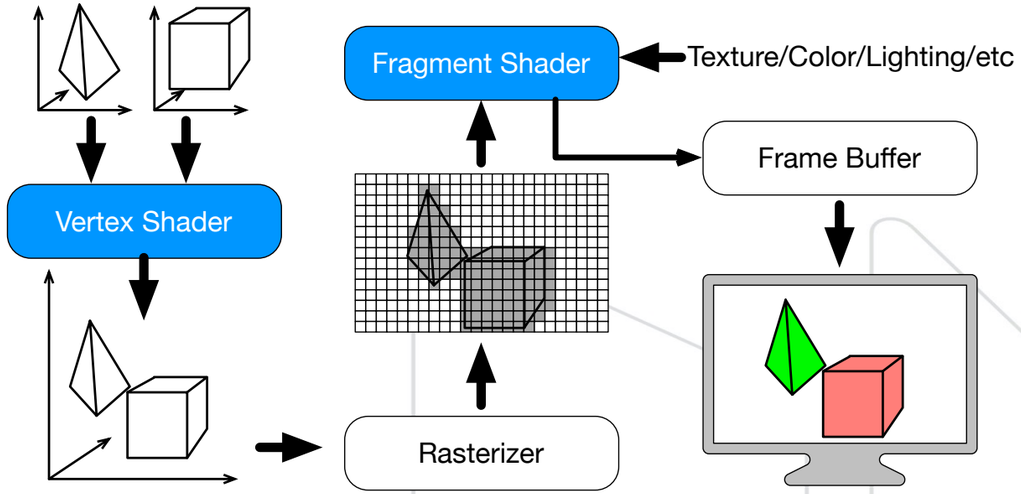
  for each primitive:

  for each sample:

  compute coverage

  compute color

  – 使用z buffer求可见图元



- 光线追踪

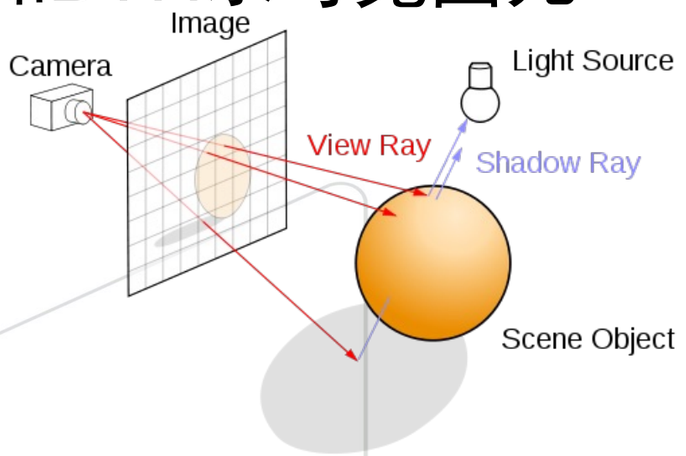
  for each sample:

  for each primitive:

  compute coverage

  compute color

  – 使用空间划分数据结构如BVH求可见图元

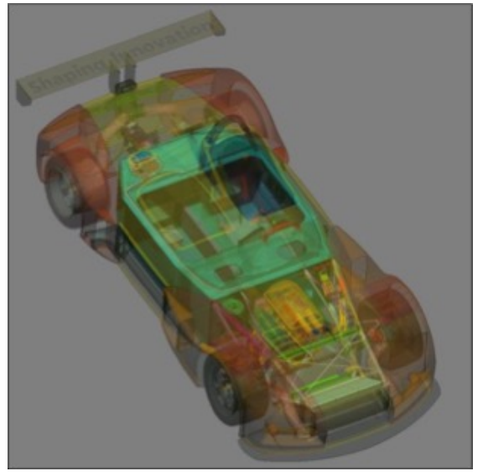


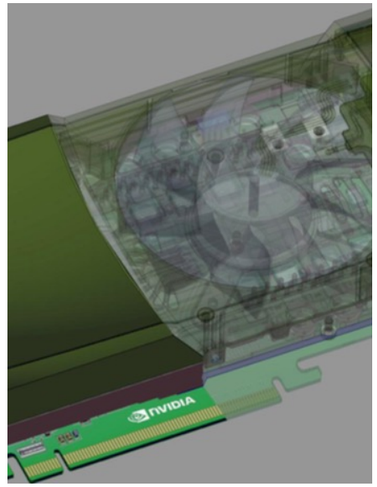
半透明场景渲染

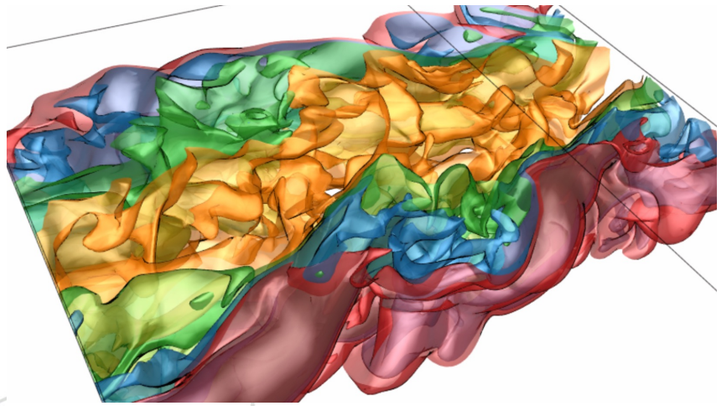
- 场景表示
  - 网格与体数据
- 网格渲染
  - 不透明：深度测试
  - 半透明：blending + order independent transparency
- 体数据渲染
  - Direct volume rendering
- 转换与融合
  - 体数据→网格：marching cubes
  - 网格→体数据：signed distance field (SDF), occupancy field
  - 神经网络渲染

包含半透明物体的场景表示

- 半透明场景
  - 包含半透明物体（如玻璃）的渲染场景
  - 需要展示被遮挡物体以研究其内部结构
    - 常见于工业制造与科学数据
  - 渲染成本极高，游戏场景中极为罕见
  - 体数据（volume）
    - 模拟数据：气象，海洋，燃烧反应等
    - 实验测量：医学影像，燃烧反应，风洞实验等
  - 网格（mesh）
    - 建模：工业制造，建筑等
    - 扫描：三维点云重建
    - 等值面：体数据中通过等值面提取得到

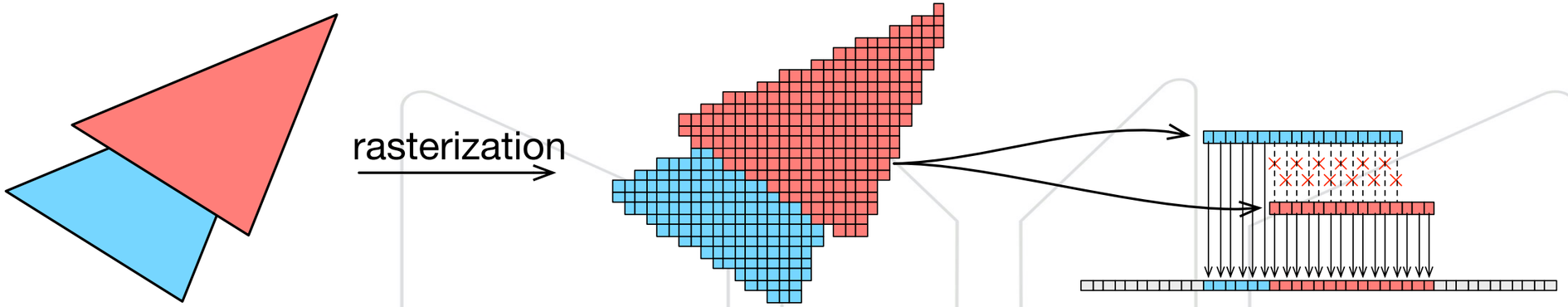


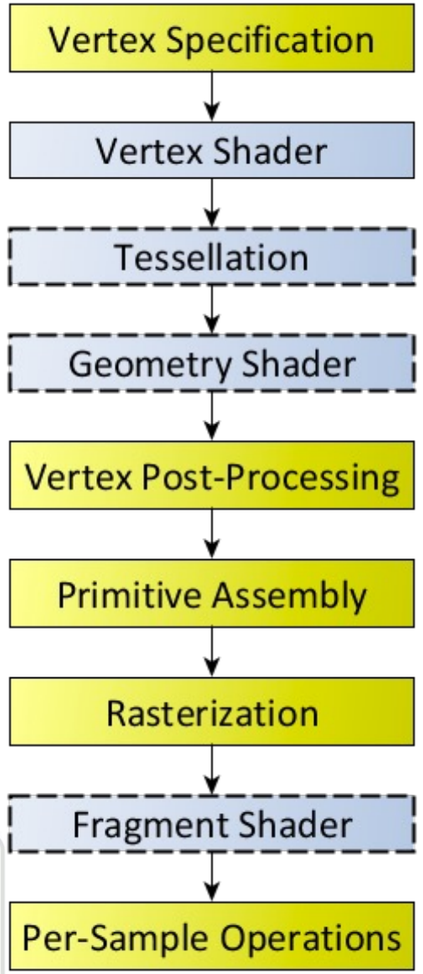




绘制不透明物体

- 不透明物体的绘制流程
  - 回顾渲染管线
    - https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview
    - 顶点→图元→片元→屏幕像素
    - Vertex shader决定如何计算顶点的屏幕坐标及其他属性
    - Fragment shader决定片元的颜色
    - 但同一个像素可能对应多个fragments
    - 如何解决冲突？





绘制不透明物体

- 深度测试
  - OpenGL默认状态（不打开深度测试）
    - Fragment直接覆盖原先像素中的内容
    - 后绘制的物体遮挡先绘制的物体
  - 打开深度测试`glEnable(GL_DEPTH_TEST)`
    - 使用深度缓存（depth buffer）保存深度信息
      - 在每次绘制前通过`glClear(GL_DEPTH_BUFFER_BIT)`清空深度缓存
      - 常与清空颜色缓存一同执行
      - `glClear(GL_COLOR_BUFFER_BIT |GL_DEPTH_BUFFER_BIT)`
    - 将fragment写入像素前先进行测试
      - 通过测试则进行写入
      - 否则将忽略fragment而不进行写入

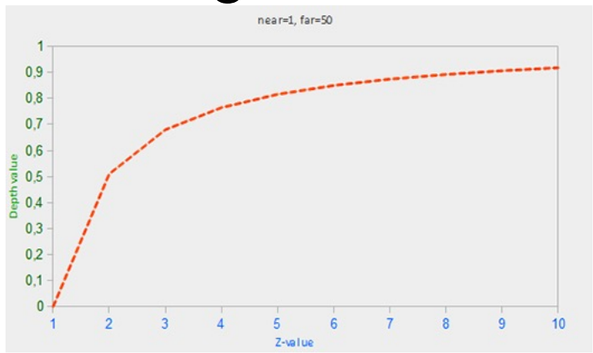
绘制不透明物体

- 深度测试
  - 打开深度测试`glEnable(GL_DEPTH_TEST)`
    - 将fragment写入像素前先进行测试
    - 默认深度测试函数为`glDepthFunc(GL_LESS)`
      - fragment深度小于depth buffer中的深度则通过测试
      - 靠近视点的fragment通过测试，遮挡较远的物体
      - 其他选项包括GL_ALWAYS, GL_NEVER, GL_EQUAL, GL_LEQUAL, GL_GREATER, GL_NOTEQUAL, GL_GEQUAL
      - GL_ALWAYS（永远通过测试）与`glEnable(GL_DEPTH_TEST)`效果相同
    - 只读depth buffer：`glDepthMask(GL_FALSE)`
      - 只读取depth buffer并进行测试，但不对depth buffer进行更新
      - 例如，当渲染场景中同时包括半透明与不透明物体时，可先打开深度测试绘制不透明物体（只绘制最靠近视点的一层），而在绘制不透明物体时，则需要绘制所有不透明物体前方的物体（只进行测试，而不更新depth buffer）

绘制不透明物体

- 深度测试
  - 可视化depth buffer
    - 可将depth buffer取出，作为纹理贴图至长方形上进行显示
    - 或重写fragment buffer，在绘制时使用深度决定fragment颜色
      - 注意gl_FragCoord.z与深度间为非线性关系

$$» F_{depth} = \frac{\frac{1}{z} - \frac{1}{near}}{\frac{1}{far} - \frac{1}{near}}$$

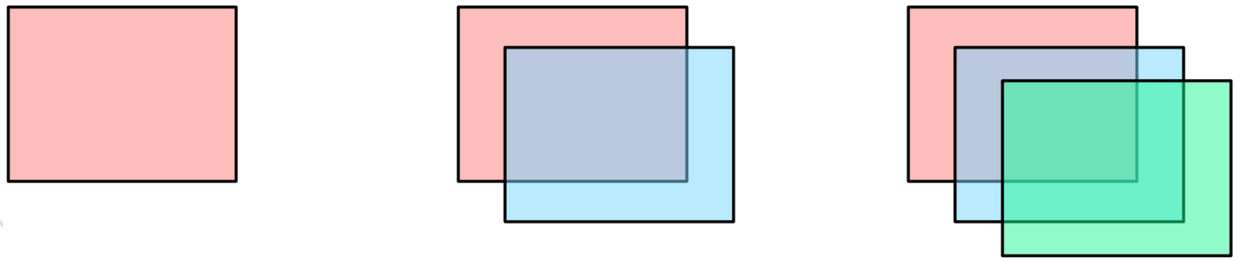


      - 在绘制前需要先对其线性化

```
float LinearizeDepth(float depth) {
    float z = depth * 2.0 - 1.0; // back to NDC
    return (2.0 * near * far) / (far + near - z * (far - near));
}
void main(){
    // divide by far for demonstration
    float depth = LinearizeDepth(gl_FragCoord.z) / far;
    FragColor = vec4(vec3(depth), 1.0);
}
```

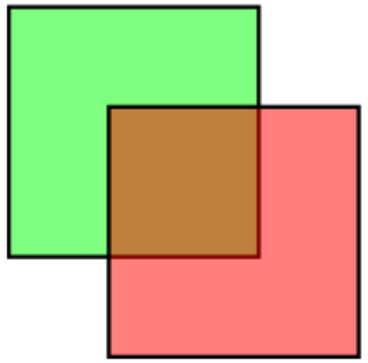
绘制半透明物体

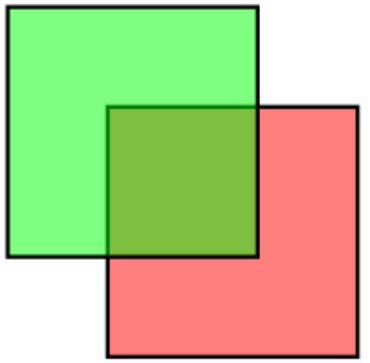
- 融合多层半透明曲面
  - 打开blending功能`glEnable(GL_BLEND)`
  - 决定blending函数`glBlendFunc(sFactor,dFactor)`
    - 最常用blending函数：
    - `glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA)`
    - $\boldsymbol{c} = \alpha_s \boldsymbol{c}_s + (1 - \alpha_s)\boldsymbol{c}_d$
    - 适用于从远至近将半透明曲面一层层叠加
      - 初始：c为红色
      - 叠加蓝色：$c = 0.5$蓝$ + 0.5c$
      - 叠加绿色：$c = 0.5$绿$ + 0.5c$

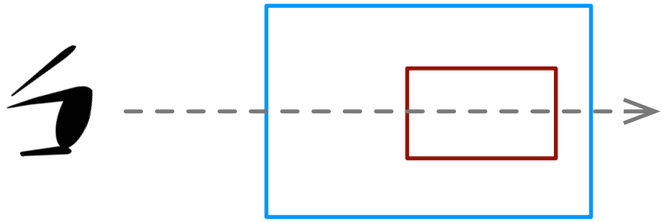


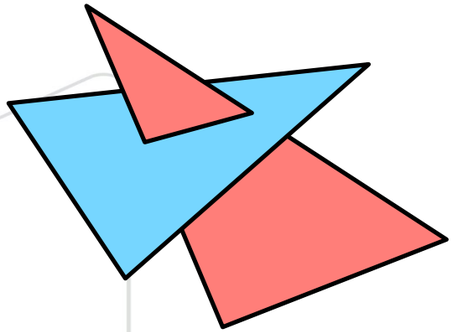
绘制半透明物体

- 融合多层半透明曲面
  - 绘制顺序很重要！
    - Blending操作不满足交换律
    - 与blending函数相关
    - 在前述函数下，需从远至近绘制
  - 如何保证绘制顺序？
    - 对物体进行深度排序无法产生精确绘制结果
    - 对三角形进行排序也无法产生精确结果
      - 三角形可能相交
      - 如何排序？平均深度？不相交亦可能产生不正确的结果
      - 运算开销大：不重叠的三角形依然会排序（不必要）
    - 需要对fragment进行排序！







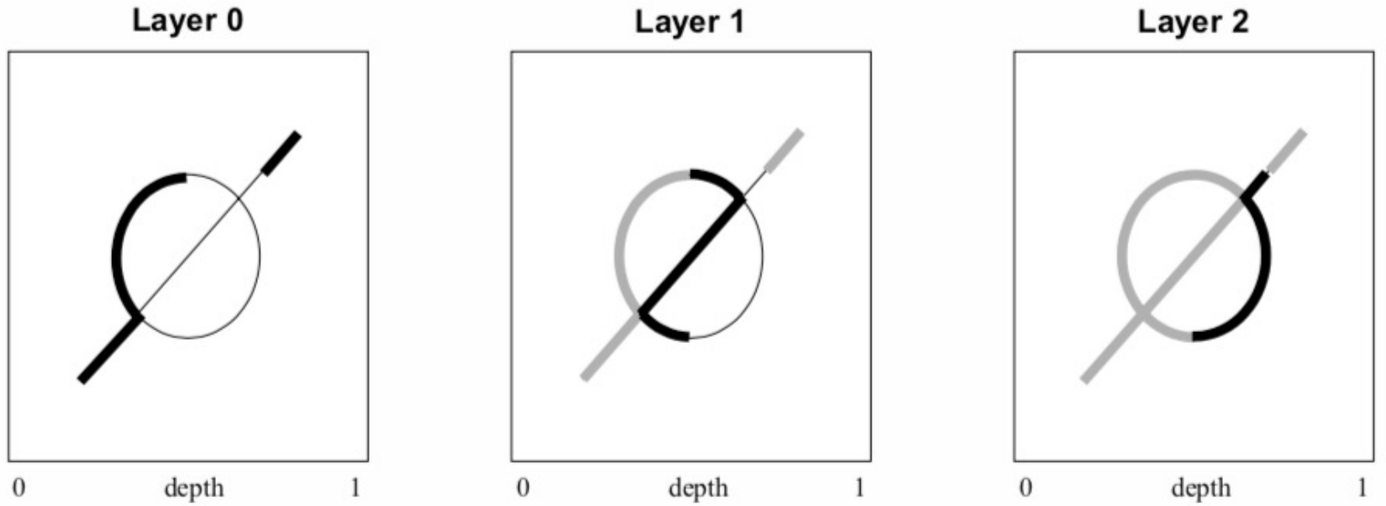


Order-Independent Transparency

- 渲染半透明曲面比体数据更复杂
  - Ray casting时，求体数据中对应采样点的值相对容易
    - 坐标转换，对三维图像插值（访问三维数组/纹理）
  - 对半透明曲面进行ray casting更为复杂
    - 需要求出每道光线所穿过的所有曲面
    - 英特尔开发的OSPray库中使用多核CPU加速光线与曲面求交
    - NVIDIA的RTX架构使用GPU加速光线与曲面求交
  - 将曲面数据先光栅化填充至一个volume中
    - 可直接使用direct volume rendering进行绘制
  - 其他方法：在此我们介绍两种基于渲染管线的方法
    - Depth peeling/dual depth peeling
    - Per-pixel linked list

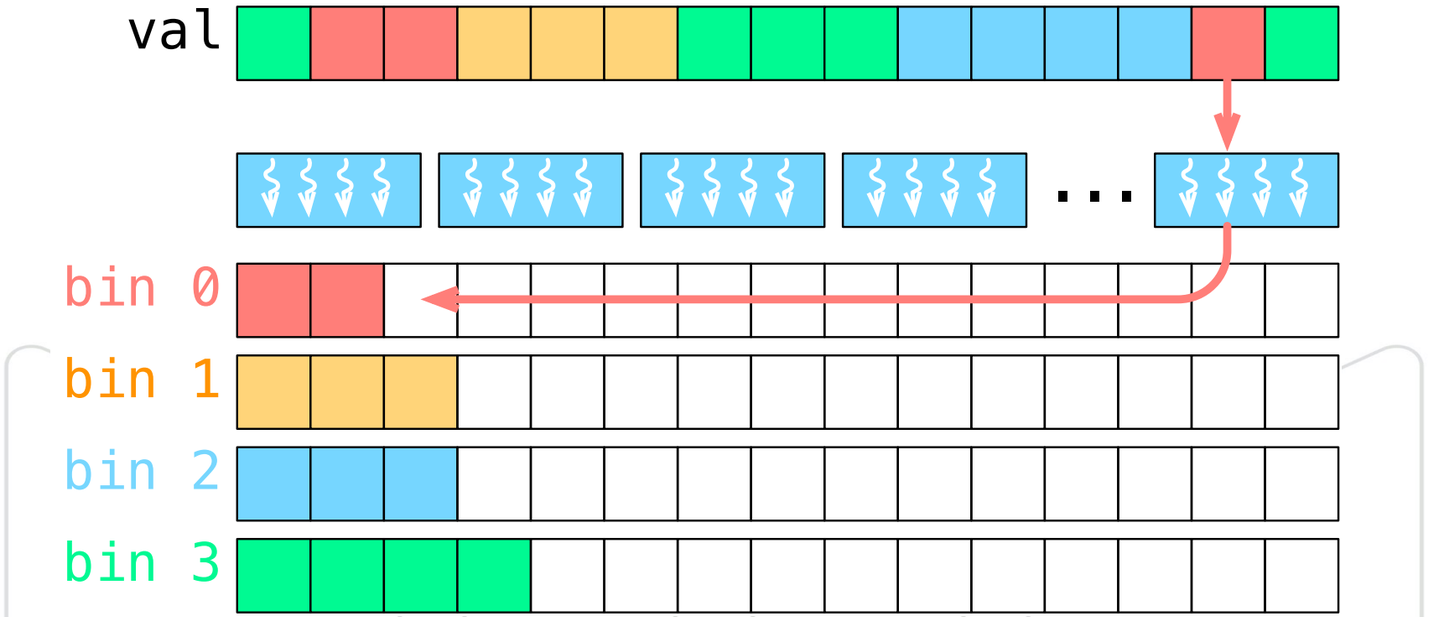
Depth Peeling

- 基本思路
  - 将fragments一层层从场景中剥除（peel）并填入至渲染结果中
    - 常被称为implicit sort：每次取出当前最近fragment（比照冒泡排序）
    - 需要两个depth buffer，一个使用GL_LESS（取最近）一个使用GL_GREATER（防止重复取值）
    - 存在问题：需要渲染多次才能得到完整场景
      - 改进版本：dual depth peeling，同时剥除最近及最远层



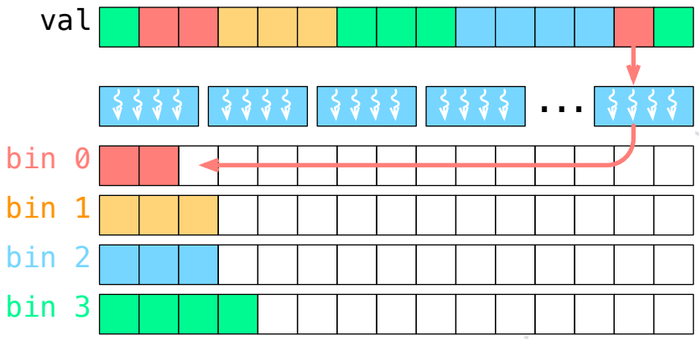
Per-pixel Link List

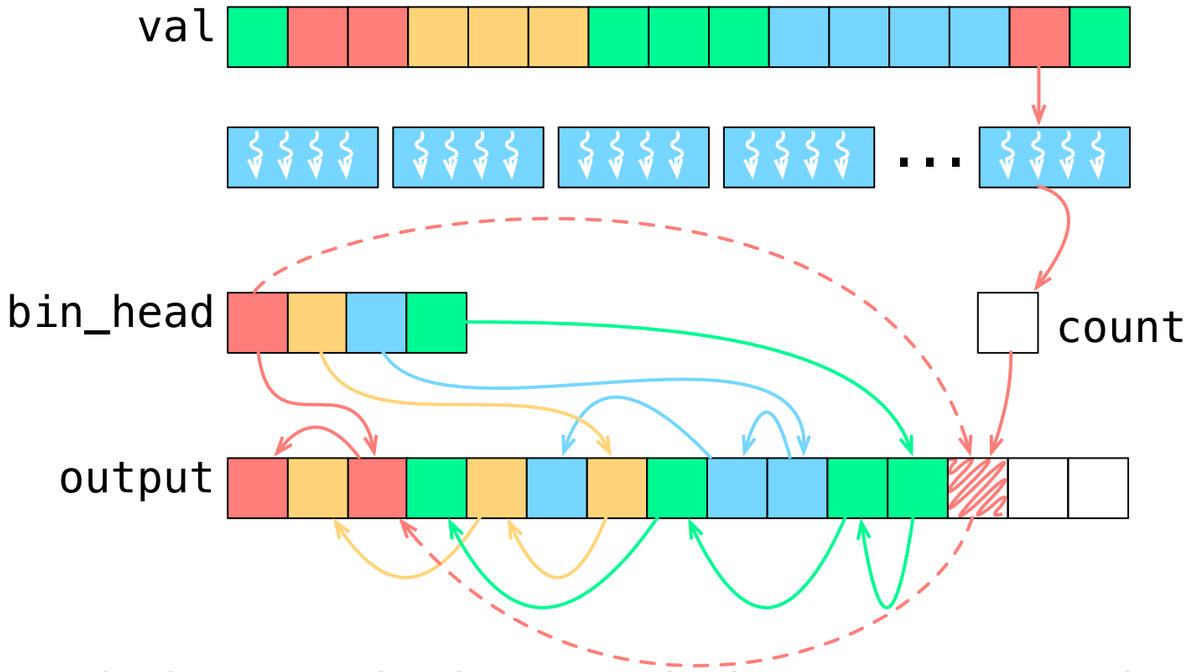
- 基本思路
  - 记录每个像素对应的所有fragments，并对其进行排序
    - 利用fragment shader并行地将fragment写入buffer中（不通过CPU）
  - 对排序后的fragments进行blending
  - 直接做法：为每个像素分配固定大小的存储空间



Per-pixel Link List

- 基本思路
  - 直接做法：为每个像素分配固定大小的存储空间
    - 存在问题：需要$width \times height \times \#layers$的空间
  - 解决方案：为每个像素建立一个链表！
    - 需要使用GLSL的原子操作消除race condition！





Questions?